UNCLASSIFIED

Defense Technical Information Center Compilation Part Notice

ADP023718

TITLE: Trusted Computing Technologies for Embedded Systems and Sensor Networks

DISTRIBUTION: Approved for public release, distribution unlimited

This paper is part of the following report:

TITLE: Proceedings of the ARO Planning Workshop on Embedded Systems and Network Security Held in Raleigh, North Carolina on February 22-23, 2007

To order the complete compilation report, use: ADA485570

The component part is provided here to allow users access to individually authored sections of proceedings, annals, symposia, etc. However, the component should be considered within the context of the overall compilation report and not as a stand-alone technical report.

The following component part numbers comprise the compilation report:
ADP023711 thru ADP023727

UNCLASSIFIED

Trusted Computing Technologies for Embedded Systems and Sensor Networks

Adrian Perrig
Carnegie Mellon University

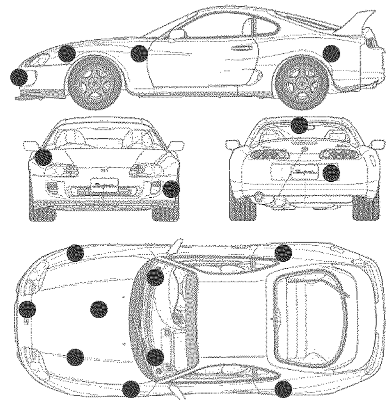
Motivation

- Embedded processors closely integrated into the fabric of everyday life
  - Anything with a powerplug is likely to already be equipped with an embedded processor
  - Additional battery-operated embedded devices are emerging (e.g., thermometers)
- Embedded processors enable new features
  - Unfortunately, features increase complexity
- Steady increase in complexity results in bugs, which require software updates to fix
- **Scary: Embedded systems with network access and code update features**

CyLab
www.cylab.cmu.edu
Carnegie Mellon

Example: Vehicular Embedded Networks

- Technology trends
  - Steady increase in number and complexity of processing units
    - GPS, in-car entertainment, safety systems
  - Car communication systems
    - DSRC, cellular technologies, BlueTooth, USB
- Security challenges:
  - Vehicular malware!



CyLab
www.cylab.cmu.edu
Carnegie Mellon

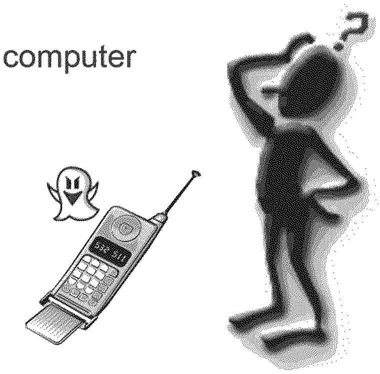
Challenges

- Ensure integrity of code executing on embedded device
  - Ensure result obtained was created by correct code
- Secure code updates
- Recovery after attack
  - Re-establish code integrity
  - Re-establish secret and authentic keys

CyLab
www.cylab.cmu.edu
Carnegie Mellon

How can we trust our devices?

- How do we securely use (potentially) compromised devices or devices we don't trust?
  - Cell phone, PDA, or car computer



CyLab
www.cylab.cmu.edu
Carnegie Mellon

Attacker Model

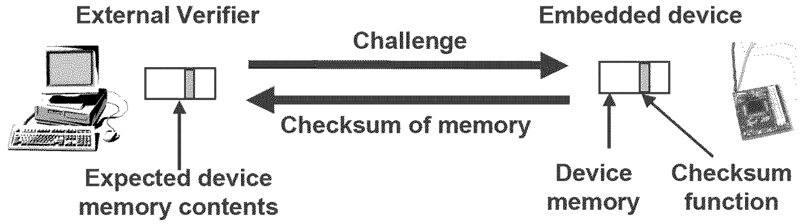
- Attacker controls software on embedded system
  - Complete control over OS, memory
  - Injection of malicious code
- No hardware modifications, verifier knows HW spec
  - Hardware attacks are much harder to perform, requires physical presence
  - Very challenging to defend against
- In this talk, assume verifier controls network, such that verified device cannot contact external helpers

Approaches to Ensure Code Integrity

- Hardware-based
  - Fixed ROM-based code
    - Cannot support code updates
  - TCG
    - Requires extra hardware, potentially high unit cost
- Software-based
  - Software-based attestation
    - Need to guard against proxy attack

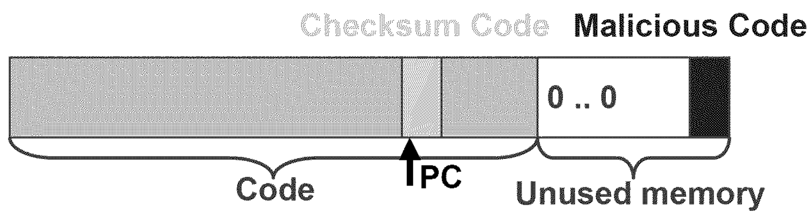
Software-based Attestation Overview

- External, trusted verifier knows expected memory content of device
- Verifier sends challenge to untrusted device
  - Assumption: attacker has full control over device's memory before check
- Device returns memory checksum, assures verifier of memory correctness



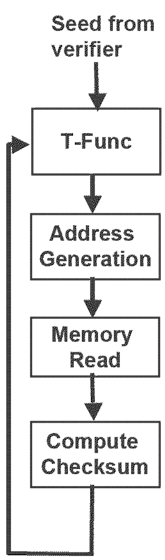
ICE: Indisputable Code Execution

- Add chksum function execution state to checksum
  - Include program counter (PC) and data pointer
- In memory copy attack, one or both will differ from original value
- Attempts to forge PC and/or data pointer increases attacker's execution time



CyLab
www.cylab.cmu.edu
Carnegie Mellon

ICE Assembler Code



Generate random number using T-Function

```
mov r15, &0x130
mov r15, &0x138
bis #0x5, &0x13A
add &0x13A, r15
```

Load byte from memory

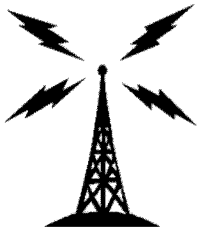
```
add r0, r6
xor @r13+, r6
```

Incorporate byte into checksum

```
add r14, r6
xor r5, r6
add r15, r6
xor r13, r6
add r4, r6
rla r4
adc r4
```

ICE Protocol

Wireless link

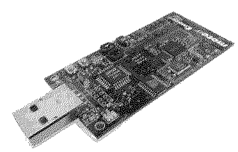


Base station

t_1: nonce, input

t_2: cksum

output



Node

- Successful verification if:
  $t_2 - t_1 <$ expected time
  and
  cksum == exp. cksum

nonce → Verf. Func. → cksum

input → Target Code → output

ICE Verification Function

- Implemented as self-checksumming code
  - Computes checksum over its own instructions
- Set up untampered execution environment
  - CPU state for atomic execution
  - E.g., turn off interrupts
- Compute checksum
  - Using memory contents and CPU state
- Checksum verifies integrity and correct set-up of execution environment

Verification Function

Target Code

CyLab
www.cylab.cmu.edu
Carnegie Mellon

ICE Properties

- Given target code T, verifier obtains property that sensor node S correctly executes T, untampered by any other (malicious) code potentially present on S
- By incorporating node ID into checksum computation, we can authenticate response

CyLab
www.cylab.cmu.edu
Carnegie Mellon

Key Establishment

- How to establish a shared secret?
  - Attacker may know entire memory contents of a newly shipped node
  - After a node has been compromised, attacker may have altered authentic public keys or knows secret keys
  - Without authentication Diffie-Hellman protocol is vulnerable to man-in-the-middle attack:
    - A → B: g^a mod p
    - B → A: g^b mod p

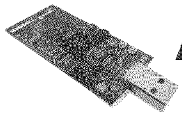
Problem Formulation

- Given nodes in a sensor network, how can any pair of nodes establish a shared secret without any prior authentic or secret information?
- In theory, this is impossible … because of active MitM attack
- Assumptions
  - Attacker cannot compute faster than sensor node
  - Each node has a unique, public, unchangeable identity stored at a fixed memory address
  - Secure source of random numbers

ICE Key Establishment

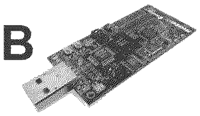
- Intuition: leverage ICE to compute checksum faster than any other node, and use that checksum as a short-lived shared secret
- Challenge: how to use short-lived shared secret to bootstrap long-lived secret?
  - Authenticate Diffie-Hellman public key

First Attempt



A

B



Pick random a

Compute $g^a \bmod p$

t_0: $g^a \bmod p$ →

Pick random b

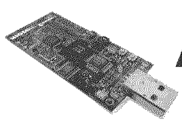
Compute $g^b \bmod p$

$g^a \bmod p$ = challenge

Compute cksum c

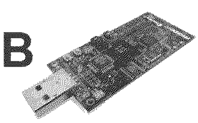
← t_1: $g^b \bmod p$, MAC(c, $g^b \bmod p$)

Second Attempt



A

B



Pick random a

Compute $g^a \bmod p$

t_0: $g^a \bmod p$ →

$g^a \bmod p$ = challenge

Compute cksum c

Pick random b

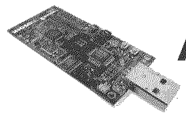
Compute $g^b \bmod p$

← t_1: $g^b \bmod p$, MAC(c, $g^b \bmod p$)

CyLab
www.cylab.cmu.edu

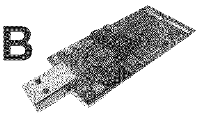
Carnegie Mellon

Guy Fawkes



A

B



Goal: A and B can authenticate each other's messages

A	B
Pick random v_2	Pick random w2
$v_1 = H(v_2)$, $v_0 = H(v_1)$	w_1=H(w_2), w_0=H(w_1)
one-way chain: $v_0 \leftarrow v_1 \leftarrow v_2$	$w_0 \leftarrow w_1 \leftarrow w_2$
Assume: A knows authentic w_0	B knows authentic v_0

$$A \xrightarrow{v_1,\ M_a,\ MAC(v_2, M_a)} B$$

$$A \xleftarrow{w_1,\ M_b,\ MAC(w_2, M_b)} B$$

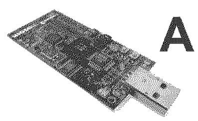
$$A \xrightarrow{v_2} B$$

$$A \xleftarrow{w_2} B$$

CyLab
www.cylab.cmu.edu

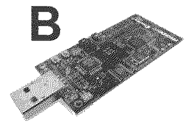
Carnegie Mellon

ICE Key Establishment



A

B



Pick random a, $g_a = g^a$ mod p

Compute $g'_a = H(g_a)$, $g''_a = H(g'_a)$, $g''_a \leftarrow g'_a \leftarrow g_a$

$$A \xrightarrow{t_0:\ g''_a} B$$

g''_a = challenge

Compute cksum c

$w_0 \leftarrow w_1 \leftarrow w_2$

$$A \xleftarrow{t_1:\ w_0\ MAC(c, w_0)} B$$

random b, g^b mod p

$$A \xrightarrow{g'_a} B$$

$$A \xleftarrow{w_1,\ g^b \bmod p,\ MAC(w_2,\ g^b \bmod p)} B$$

$$A \xrightarrow{g_a} B$$

$$A \xleftarrow{w_2} B$$

Summary: ICE Key Re-Establishment

- Protocol can prevent man-in-the-middle attacks without authentic information or shared secret
- Attacker can know entire memory content of both parties before protocol runs
- Forces attacker to introduce more powerful node into network, prevents remote attacks
- Future work: relax strong assumption that attacker cannot compute faster

Summary

- Software-based attestation provides interesting properties, but many challenges remain
  - Defeat proxy attacks in wireless environments
  - Extend properties to general computation
  - Build systems with perfect detection of code integrity attacks
  - Recover from malicious code infection
  - Provide human-verifiable guarantees
- Study use of hardware-based support
  - Determine minimal hardware requirements to provide embedded systems security